METHOD AND SYSTEM FOR SYSTEMATICALLY DIAGNOSING DATA PROBLEMS IN A DATABASE

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to data problem diagnosing tools and, more particularly, to a method and system for systematically diagnosing data problems in a database.

Description of the Related Art

DataBase Administrators (DBAs) monitor and maintain databases to ensure accurate and consistent storage of data in the databases. DBAs load and retrieve data to and from the database using a programming language called SQL (Structured Query Language) which is well known to DBAs. In a database, data are stored in tables. Each table has a set of rows and fields (columns). Each row in the table is identifiable by one or a combination of unique identifiers known as "keys" and each field in the table is identifiable by a field name. A field containing the keys is known as a "key field." DBAs and database programmers utilize keys and field names to retrieve particular data from a database table.

Conventionally, when the integrity of data stored in a database is questioned, the DBA attempts to diagnose the nature of the problem and/or the exact location of the problem using one of the standard SQL statements known as "SELECT." A SELECT statement is an SQL command for retrieving data from one or more tables of the database. Without knowing the exact nature or location of the data problem in the database, the DBA must speculate on where the problem may lie and use the SELECT statement to retrieve data from the potential problem areas of the database. The DBA compares the retrieved data with some source to determine whether inaccurate or inconsistent data are stored in the database, and whether any data is missing from a particular location in the database. This process is repeated until the DBA can diagnose properly the data problem in the database. As a result, the conventional process of diagnosing data problems in the database can be extremely time consuming and inefficient. Furthermore, the DBA must keep track of any data retrieved during this process and manually compare data sets to determine the exact nature and/or location of the data problem. This also can be tedious and is prone to human error.

Therefore, there is a need for an improved technique by which data integrity problems or other problems in a database can be diagnosed more quickly and more systematically, thereby overcoming problems encountered in conventional data problem diagnosing techniques for databases.

SUMMARY OF THE INVENTION

The present invention is directed to a method and system for systematically diagnosing data problems in a database which overcomes limitations encountered in conventional data problem diagnosing techniques for databases.

Particularly, the system of the present invention provides a test program for running simultaneously a set of tests on a database. In the test program, these tests are defined as queries. Then these queries are refined as parts of a larger query in a particular manner, such that the outcome of the queries (test results) appears merely as fields (columns) of tables. This technique can be implemented using a new SQL command, "WITH", which is defined in the SQL 1999 Standard set by ANSI/ISO (American Standards Institute / International Standards Organization). Then these fields of tables are queried again by using a well known SQL command "LEFT OUTER JOIN" on all key fields in the database to form a larger table, so that the results from the different tests can be displayed in one outcome table.

By simply running the test program and viewing the outcome table showing all the test results, an operator is able to diagnose easily the nature and location of the data problem(s) in the database.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a diagram of a system for diagnosing data problems in a database according to one embodiment of the present invention.

Figure 2 is an example of an outcome table illustrating different test results according to the present invention.

Figure 3 is a flowchart illustrating the processing steps of a method for systematically diagnosing data problems in a database according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the drawings, the same reference numerals are used to indicate the same components.

Figure 1 is a diagram of a system 100 for diagnosing data problems in a database according to one embodiment of the present invention. As shown in Fig. 1, the system 100 includes at least one database 10 and a data integrity testing module 20, operatively coupled. The database 10 includes a plurality of tables 12 for storing data therein. Each table 12 includes rows of data categorized by different fields (columns). Each row in the table is assigned to one or a combination of key values and each field is assigned to a field name. For example, a table 12 named "Shoe" stored in the database 10 has a key field 16 for identifying key values associated with each row, and fields 14 such as "Shoe Model", "Total Quantity", and "Manufacturer." By identifying the key value and the field name, a particular data record stored in the table can be accessed. For instance, by identifying the key value to be 011 and the field name to be "Shoe Model", the data record of "SH75" can be accessed from the Shoe table. One skilled in the art would readily appreciate that if the field names were

guaranteed to be unique, then they could be used as keys if desired. All this database architecture is well known in the art.

The data integrity testing module 20 includes a computer program (referred to herein as a "test program") for simultaneously running a group of different tests on the database and a microprocessor or the like for controlling and executing the test program. The test program is written in SQL. In accordance with one embodiment, this test program is written by a programmer (or DBA) at the initial set-up of the system 100. To accomplish this task, the programmer identifies a group of different tests that the programmer desires to run systematically. The different tests may include, but are not limited, to a test of comparing data in the same fields of different tables in the database if the different tables are expected to have the same value and data type (e.g., both Tables 1 and 2 may have equivalent "Shoe Model" type fields), a test of verifying that data or accurate data actually exists in particular fields/rows of the tables, and a test of counting the number of items in particular fields/rows that satisfy certain conditions.

Once the desired tests are identified, the programmer writes the program based on the identified tests. Particularly, each of the identified tests is written as an SQL query and then these queries are refined as fields of a larger query. This is implemented using an SQL command "WITH", the use of which will be described below in more detail. The test program is also written so that it generates automatically a table identifying all the results from the different tests. This table is referred to herein as the "outcome" table. The generation of the outcome table can be implemented using a well-known SQL command "LEFT OUTER JOIN", the use of which will also be

described below in more detail.

In response to instructions from a user such as the DBA or whenever data testing is desired for the database 10, the data integrity testing module 20 is configured to execute the test program on the database 10. The testing module 20 generates automatically the outcome table identifying all the different test results at one time. The outcome table allows the user to quickly understand the different test results and to identify the nature and/or location of data problems in the database.

In accordance with another embodiment, instead of predefining the test program in the system at the initial set-up of the system, the system is configured so that the user can select a set of tests to run on the database. That is, the user can formulate and run his preferred test program, instead of the predefined test program. This can be accomplished as follows. A user interface 22 (represented in dotted lines in Fig. 1), which is coupled to the data integrity testing module 20, is provided in the system 100. The user interface includes a display device, an input unit, a processor, etc., for communicating user inputs to the testing module 20. The user interface 22 is configured to display (e.g., on the display device) a list of predetermined tests from which the user can select. The user selects one or more of the desired tests from the list using, e.g., the input unit such as a keyboard.

Then, based on the selected tests, predetermined questions are displayed to the user under control of the testing module 20. These questions may be about the relationship between various tables in the database, the fields or keys of the tables, expected values in these fields, conditions under which these tests should be run, and

any other information pertaining to the database. These questions are predetermined and designed to obtain information needed to prepare a test program for running those tests selected by the user. Based on the user's input, the data integrity testing module 20 or some other component coupled to the system 100 generates a test program. This can be accomplished by providing the testing module 20 or other component with existing computer software for automatically generating the appropriate SQL syntax based the received user input and certain criteria. The generated test program is then stored in the testing module 20 and executed whenever desired as discussed above. It must be noted, however, that there is a performance tradeoff between the number of tests performed and the time required to execute these tests.

Fig. 2 is one example of an outcome table displaying the results of different tests that may be performed by the data integrity testing module 20 according to one embodiment of the present invention. In this example, the entire rows of data were tested using five different tests, Tests 1, 2, 3, 4, and 5. As shown in Fig. 2, the results of the different tests may be indicated using an indicator such as an "X" under appropriate test names in connection with tested rows identified by key values. Here, the presence of the indicator "X" represents a test failure and the lack of "X" represents a test passed. For example, in this outcome table, the row entry identified by the key value of "011" is marked with "X" which indicates that this row has failed Test 1. The test failure may indicate a number of different things depending on the nature of the tests. For example, if the test was to detect data records that are obviously out of range, then the failure of this test would indicate that the particular data record is out

of range and should be corrected. If the test is to determine whether any field in the row identified by the key value is missing entirely, the failure of this test would mean that there is no actual data stored in some field of that row.

In accordance with the present invention, the use of the SQL syntax, "WITH" and "LEFT OUTER JOIN", is now described. A WITH command is a relatively new SQL command and allows several queries to appear as one large query. The WITH command is defined in the SQL 1999 Standard set by ANSI/ISO (American National Standards Institute / International Standards Institute) and is described therein also as a "recursive select." It is used in databases such as DB2 and Ingres. In the present invention, a set of tests are defined as small queries. Then these queries are redefined as parts of a larger query using the WITH command so that the outcomes of the queries (test results) appear as fields (columns) of tables. The following is one example of high level pseudo-SQL syntax for redefining test queries as part of a larger query where the outcome of each query is declared as fields of a table as discussed above:

```
L1:  WITH
L2:    Test1 (field1, field2) as (<query1>)
L3:   ,Test2 (field1, field2) as (<query2>)
L4:  SELECT
L5:    field1
L6:   ,field2
L7:  FROM
L8:    Test1
```

L9: ,Test2

L10: WHERE <conditions>...

It should be understood that one skilled in SQL would readily appreciate the operation implemented by the above SQL syntax. For instance, lines L2 and L3 declare each small queries "query1" and "query2" (representing tests) which produce the fields "field1" and "field2" for each of two temporary tables named "Test1" and "Test2." In theory, there can be any number of these tests, subject to the performance constraints mentioned above. Each test, however, needs to define fields of the same name and type. This allows the outcome of the queries to be represented in these temporary tables. For each table "Test 1" and "Test 2," one of the fields "field1" and "field2" is a key field and the other field is an indicator field having certain values/characters for indicating test results. For example, if a failed test result is to be represented using an indicator "X" as in Fig. 2, then the indicator field will have either the "X" character or a NULL value for indicating the fail/pass test results. The WITH command at line L1 then allows the statements on lines L4 - L10 to use the two tables Test1 and Test2 produced from the execution of lines L2 and L3 to form another query which is combined with the use of "LEFT OUTER JOIN" commands (not shown above) between the set of all keys and the set of keys returned for each test.

A LEFT OUTER JOIN command is a well-known SQL command which is used to align and associate certain fields. Here, for each field (e.g., 2, 3, 4 ...n) of the test results, the LEFT OUTER JOIN command is used (e.g., in line L10) to align the keys of the results of each test performed with the set of all key values, so that, if desired, all keys are displayed in the first column of the outcome table, followed by columns

indicating the test results with the use of an indicator such as "X" as in Fig. 2. One skilled in SQL syntax would understand the implementation and use of the LEFT OUTER JOIN command in the context described above to produce these desired results. For instance, one of the tables "Test1" and "Test 2" would be produced by using a well known SQL command "SELECT" on all keys so that these tables can be used as the basis for using the LEFT OUTER JOIN command. In general, the first table in a LEFT OUTER JOIN (literally, the one on the left) is a superset which drives the number of rows in a test results set. In this example, columns to the right of this first table then will have the indicator value of either "X" or a NULL value.

The use of the WITH and LEFT OUTER JOIN commands allows the results of queries of underlying database tables or views, which are not constrained to have the same field parameters (the same field name and data type), to be declared as similar tables Test1, Test2, ... Testn. Since these query results are then stored in tables having the same field parameters (field name and data type), a larger query can be written based on these results. That is, Test1, Test2, ... Testn, gather raw information and set it all into the same context so that a larger query (lines L4-L10) can compare them all at once. In other embodiments, other SQL's OUTER JOIN syntaxes can be used as well in lieu of the LEFT OUTER JOIN command depending on the application and need.

Because it is often desirable to have an outcome table showing only those rows in database tables that have failed one or more tests (as shown in Fig. 2), constraints can be imposed (e.g., in line L10) so that at least one of the columns 2 through n in the outcome table have a non-NULL value.

The following is another example of high level pseudo-SQL syntax for running two tests simultaneously. Here, however, the tests are performed on one particular known key value although a set of known keys can be tested as well. This syntax is identical to the one shown above, except that the queries are executed by setting the condition (using "WHERE") that a key equals a known value. One skilled in SQL would readily appreciate the operation implemented by the below SQL syntax and thus, it will not be described further:

```
WITH
Test1 (field1, field2) as (
  <query1
    WHERE <key> = <known value>
  >
),
Test2 (field1, field2) as (
  <query2
    WHERE <key> = <known value>
  >
)
SELECT
  field1, field2
FROM
  Test1, Test2
WHERE <conditions>.
```

Fig. 3 is a flowchart illustrating the processing steps of a method for diagnosing data problems in a database according to one embodiment of the present invention. These processing steps can be implemented by the system 100 of Fig. 1. As shown in Fig. 3, in Step S2, a set of tests to be performed are identified by a DBA, operator or any other user who desires to run the tests on the database. In Step S4, a test program is prepared, e.g., by combining different test queries as a larger query using the WITH and LEFT OUTER JOIN commands in SQL based on the identified set of

tests, as discussed above. The test program may be written manually by a programmer, or automatically by computer software. Also, Steps S2 and S4 can be performed at the initial setup of the system 100 or before the user is ready to run the tests. The test program is stored in the data integrity testing module 20 in the system 100.

In Step S6, the data integrity testing module 20 runs the test program on the database according to instructions from the user or whenever the testing of the database is desired. In Step S8, the results from all different tests are combined and displayed in an outcome table by running the test program. The user can view the outcome table to diagnose easily the nature and location of any problem in the database.

The outcome table can be displayed via any number of platforms. For example, the outcome table can be displayed on a web page of a particular website, or may be printed automatically on paper. The outcome table may be displayed using a display device of a handheld, laptop, or any computer device capable of communicating with the data integrity testing module 20.

The present invention contemplates as part of the invention that the use of the SQL commands "WITH" and "LEFT OUTER JOIN" may be substituted by using other programming techniques as long as the tests can be queried as parts of a larger query to display an outcome table.

Accordingly, the present invention provides a user with an effective and powerful problem diagnosing tool for databases. By simply running the test program and viewing the outcome table, the user can easily diagnose the nature and location of data

problems in a database.

The invention being thus described, it will be obvious that the same may be varied in many ways. Such variations are not to be regarded as a departure from the spirit and scope of the invention, and all such modifications as would be obvious to one skilled in the art are intended to be included within the scope of the following claims.